Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования
Кафедра «Математической кибернетики и информационных технологий»

Отчет о проекте
по дисциплине “Введение в ИТ”
на тему:
“HTTP-приложение Akka”

Выполнил: студент БВТ1903
Нкурикийе Хафидати
Проверила:
Мосева Марина Сергеевна

Москва 2021

Оглавление

Цель работы

Цель работы заключается в запуске и тестировании HTTP-приложения Akka, получении предварительного обзора того, как маршруты упрощают обмен данными по HTTP.

Приложение должно быть реализовано в следующих четырех исходных файлах:

• QuickstartApp.scala - содержит основной метод начальной загрузки приложения.

• UserRoutes.scala - HTTP-маршруты Akka, определяющие открытые эндпоинты.

• UserRegistry.scala - актор, обрабатывающий запросы на регистрацию.

• JsonFormats.scala - преобразует данные JSON из запросов в типы Scala и из типов Scala в ответы JSON.

Ход работы

Создаем новый проект в IDE IntelliJ IDEA для языка Scala, на основе sbt.

После успешного создания базового проекта настроим параметры сборки в файле *build.sbt* следующим образом.

```
lazy val akkaHttpVersion = "10.2.7"
lazy val akkaVersion = "2.6.17"
lazy val root = (project in file(".")).
  settings(
    inThisBuild(List(
      organization := "com.example",
      scalaVersion := "2.13.4"
    )),
    name := "akka-http-quickstart-scala",
    libraryDependencies ++= Seq(
      "com.typesafe.akka" %% "akka-http" % akkaHttpVersion,
      "com.typesafe.akka" %% "akka-http-spray-json" % ak-
kaHttpVersion,
      "com.typesafe.akka" %% "akka-actor-typed" % akkaVersion,
      "com.typesafe.akka" %% "akka-stream" % akkaVersion,
      "ch.qos.logback" % "logback-classic" % "1.2.3",
      "com.typesafe.akka" %% "akka-http-testkit" % akkaHttpVer-
sion %
        Test,
      "com.typesafe.akka" %% "akka-actor-testkit-typed" % ak-
kaVersion %
        Test,
      "org.scalatest" %% "scalatest" % "3.1.4" %
        Test
```

```
    )
  )
```

рисунок 1. Содержимое файла build.sbt

```
my-app {
  routes {
    # If ask takes more time than this to complete the request is
failed
    ask-timeout = 5s
  }
}
```

рисунок 2. Содержимое файла application.conf

```
<configuration>
    <!-- This is a development logging configuration that logs to
standard out, for an example of a production
        logging config, see the Akka docs:
https://doc.akka.io/docs/akka/2.6/typed/logging.html#logback -->
    <appender name="STDOUT" target="System.out"
class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>[%date{ISO8601}] [%level] [%logger]
[%thread] [%X{akkaSource}] - %msg%n</pattern>
        </encoder>
    </appender>

    <appender name="ASYNC" class="ch.qos.logback.classic.AsyncAp-
pender">
        <queueSize>1024</queueSize>
        <neverBlock>true</neverBlock>
        <appender-ref ref="STDOUT" />
    </appender>

    <root level="INFO">
        <appender-ref ref="ASYNC"/>
    </root>

</configuration>
```

рисунок 3. Содержимое файла logback.xml

```scala
package com.example

import akka.actor.typed.ActorRef
import akka.actor.typed.Behavior
import akka.actor.typed.scaladsl.Behaviors
import scala.collection.immutable

final case class User(name: String, age: Int, countryOfResidence:
String)
final case class Users(users: immutable.Seq[User])

object UserRegistry {
  sealed trait Command
  final case class GetUsers(replyTo: ActorRef[Users]) extends
Command
  final case class CreateUser(user: User, replyTo: ActorRef[Ac-
tionPerformed]) extends Command
  final case class GetUser(name: String, replyTo: Actor-
Ref[GetUserResponse]) extends Command
  final case class DeleteUser(name: String, replyTo: ActorRef[Ac-
tionPerformed]) extends Command

  final case class GetUserResponse(maybeUser: Option[User])
  final case class ActionPerformed(description: String)

  def apply(): Behavior[Command] = registry(Set.empty)

  private def registry(users: Set[User]): Behavior[Command] =
    Behaviors.receiveMessage {
      case GetUsers(replyTo) =>
        replyTo ! Users(users.toSeq)
        Behaviors.same
      case CreateUser(user, replyTo) =>
        replyTo ! ActionPerformed(s"User ${user.name} created.")
        registry(users + user)
      case GetUser(name, replyTo) =>
        replyTo ! GetUserResponse(users.find(_.name == name))
        Behaviors.same
      case DeleteUser(name, replyTo) =>
        replyTo ! ActionPerformed(s"User $name deleted.")
        registry(users.filterNot(_.name == name))
    }
}
```

рисунок 4. Содержимое файла UserRegistry.scala

```
package com.example

import akka.actor.typed.ActorSystem
import akka.actor.typed.scaladsl.Behaviors
import akka.http.scaladsl.Http
import akka.http.scaladsl.server.Route

import scala.util.Failure
import scala.util.Success

// Основной класс
object QuickstartApp {
  // Запуск HTTP сервака
  private def startHttpServer(routes: Route)(implicit system: Ac-
torSystem[_]): Unit = {
    // Akka HTTP нуждается в классическом ActorSystem для запуска
    import system.executionContext

    val futureBinding = Http().newServerAt("localhost",
8080).bind(routes)
    futureBinding.onComplete {
      case Success(binding) =>
        val address = binding.localAddress
        system.log.info(s"Server online at http://${ad-
dress.getHostString}:${address.getPort}/")
      case Failure(ex) =>
        system.log.error(s"Failed to bind HTTP endpoint, termi-
nating system ${ex}")
        system.terminate()
    }
  }
  def main(args: Array[String]): Unit = {
    val rootBehavior = Behaviors.setup[Nothing] { context =>
      val userRegistryActor = context.spawn(UserRegistry(),
"UserRegistryActor")
      context.watch(userRegistryActor)

      val routes = new UserRoutes(userRegistryActor)(context.sys-
tem)
      startHttpServer(routes.userRoutes)(context.system)

      Behaviors.empty
    }
    val system = ActorSystem[Nothing](rootBehavior, "HelloAk-
kaHttpServer")
  }
}
```

рисунок 5. Содержимое файла QuickstartApp.scala

```
package com.example

import akka.http.scaladsl.server.Directives._
import akka.http.scaladsl.model.StatusCodes
import akka.http.scaladsl.server.Route

import scala.concurrent.Future
import com.example.UserRegistry._
import akka.actor.typed.ActorRef
import akka.actor.typed.ActorSystem
import akka.actor.typed.scaladsl.AskPattern._
import akka.util.Timeout

class UserRoutes(userRegistry: ActorRef[UserRegistry.Com-
mand])(implicit val system: ActorSystem[_]) {

  import akka.http.scaladsl.marshallers.sprayjson.SprayJsonSup-
port._
  import JsonFormats._

  private implicit val timeout = Timeout.create(system.set-
tings.config.getDuration("my-app.routes.ask-timeout"))

  def getUsers(): Future[Users] =
    userRegistry.ask(GetUsers)
  def getUser(name: String): Future[GetUserResponse] =
    userRegistry.ask(GetUser(name, _))
  def createUser(user: User): Future[ActionPerformed] =
    userRegistry.ask(CreateUser(user, _))
  def deleteUser(name: String): Future[ActionPerformed] =
    userRegistry.ask(DeleteUser(name, _))

  val userRoutes: Route =
    pathPrefix("users") {
      concat(
        pathEnd {
          concat(
            get {
              complete(getUsers())
            },
            post {
              entity(as[User]) { user =>
                onSuccess(createUser(user)) { performed =>
                  complete((StatusCodes.Created, performed))
                }
              }
            })
        },
        path(Segment) { name =>
          concat(
            get {
              rejectEmptyResponse {
                onSuccess(getUser(name)) { response =>
```

```
                    complete(response.maybeUser)
                  }
                }
              },
              delete {
                onSuccess(deleteUser(name)) { performed =>
                  complete((StatusCodes.OK, performed))
                }
              })
          })
      }
}
```

рисунок 5. Содержимое файла UserRoutes.scala

```
package com.example

import com.example.UserRegistry.ActionPerformed

import spray.json.DefaultJsonProtocol

object JsonFormats  {
  // Импорт классического кодеровщика в примитивные типы Scala
(Int, List, String ...)
  import DefaultJsonProtocol._

  implicit val userJsonFormat = jsonFormat3(User)
  implicit val usersJsonFormat = jsonFormat1(Users)

  implicit val actionPerformedJsonFormat = jsonFormat1(ActionPer-
formed)
}
```

рисунок 6. Содержимое файла JsonFormats.scala

Приложение реализовано в следующих четырех исходных файлах:

- *QuickstartApp.scala* — содержит основной метод начальной загрузки приложения.
- *UserRoutes.scala* — HTTP-маршруты Akka, определяющие открытые эндпоинты.
- *UserRegistry.scala* — актор, обрабатывающий запросы на регистрацию.
- *JsonFormats.scala* — преобразует данные JSON из запросов в типы Scala и из типов Scala в ответы JSON.

Пример модульного теста:

```
import akka.actor.testkit.typed.scaladsl.ActorTestKit
import akka.http.scaladsl.marshalling.Marshal
import akka.http.scaladsl.model._
import akka.http.scaladsl.testkit.ScalatestRouteTest
import org.scalatest.concurrent.ScalaFutures
import org.scalatest.matchers.should.Matchers
import org.scalatest.wordspec.AnyWordSpec

class UserRoutesSpec extends AnyWordSpec with Matchers with
ScalaFutures with ScalatestRouteTest {

  lazy val testKit = ActorTestKit()
  implicit def typedSystem = testKit.system
  override def createActorSystem(): akka.actor.ActorSystem =
    testKit.system.classicSystem

  val userRegistry = testKit.spawn(UserRegistry())
  lazy val routes = new UserRoutes(userRegistry).userRoutes

  import
akka.http.scaladsl.marshallers.sprayjson.SprayJsonSupport._
  import JsonFormats._
  "UserRoutes" should {
    "return no users if no present (GET /users)" in {
      val request = HttpRequest(uri = "/users")

      request ~> routes ~> check {
        status should ===(StatusCodes.OK)

        contentType should ===(ContentTypes.`application/json`)

        entityAs[String] should ===("""{"users":[]}""")
      }
    }
    "be able to add users (POST /users)" in {
      val user = User("Kapi", 42, "jp")
      val userEntity = Marshal(user).to[MessageEntity].fu-
tureValue // futureValue is from ScalaFutures

      val request = Post("/users").withEntity(userEntity)

      request ~> routes ~> check {
        status should ===(StatusCodes.Created)

        contentType should ===(ContentTypes.`application/json`)

        entityAs[String] should ===("""{"description":"User Kapi
created."}""")
      }
    }
    "be able to remove users (DELETE /users)" in {
      val request = Delete(uri = "/users/Kapi")
```

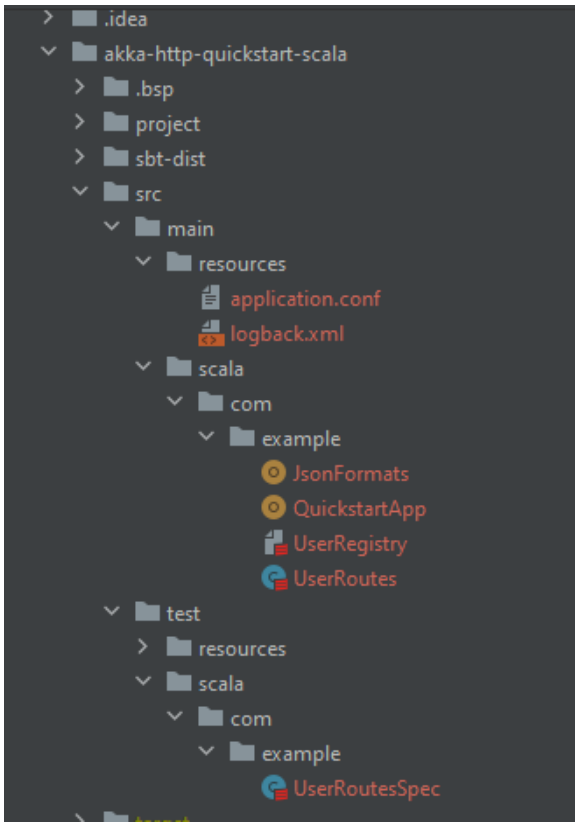
```
      request ~> routes ~> check {
        status should ===(StatusCodes.OK)

        contentType should ===(ContentTypes.`application/json`)

        entityAs[String] should ===("""{"description":"User Kapi
deleted."}""")
      }
    }
  }
}
```

рисунок 7. Код unit теста

Файлы расположены согласно скриншоту ниже.



Рисинок 8 - расположение файлов

Запуск и тестирование приложения

Запускаем проект и ждем появления в открывшейся консоли сообщения “… Server online at …”

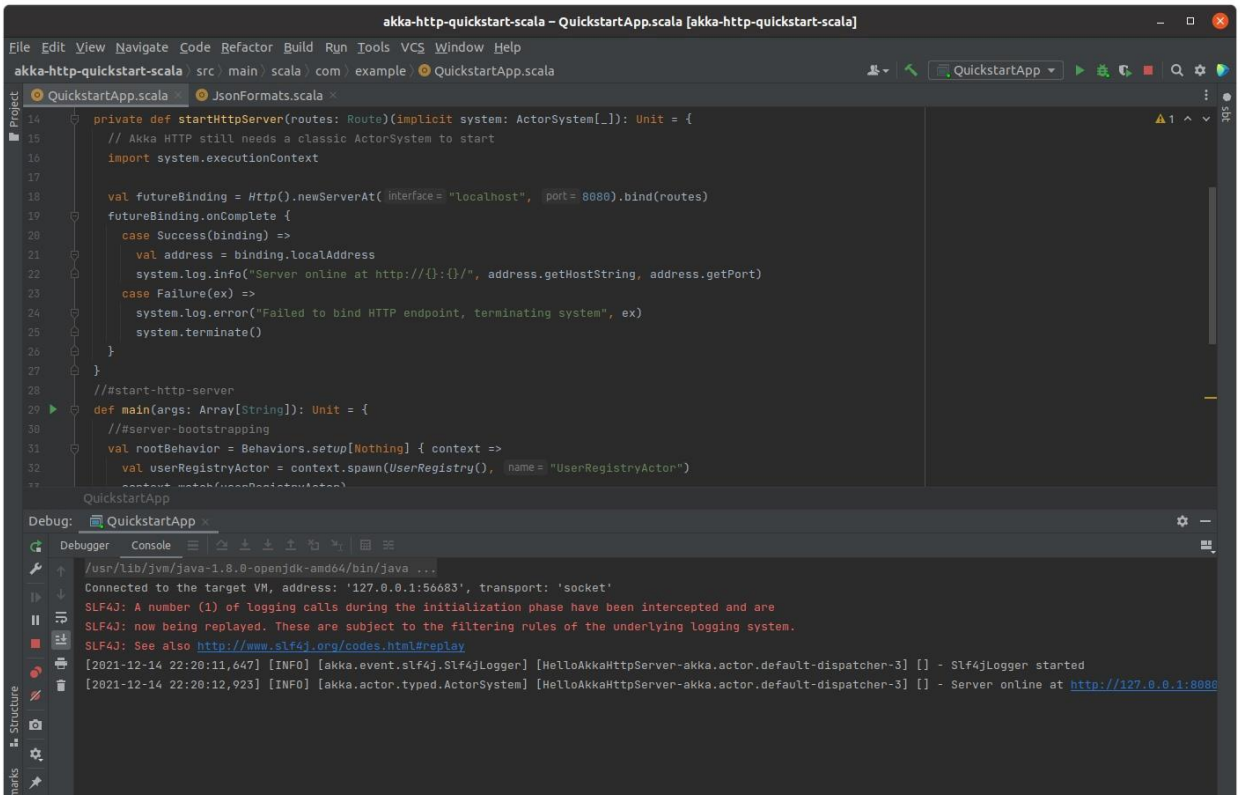


рисунок 9. Снимок логирования только что запущенного приложения

Теперь протестируем наше HTTP приложения с помощью инструмента Postman.

Отправим POST запрос нашему приложению поместив в тело запроса в формате JSON параметры пользователя, которого мы хотим добавить.

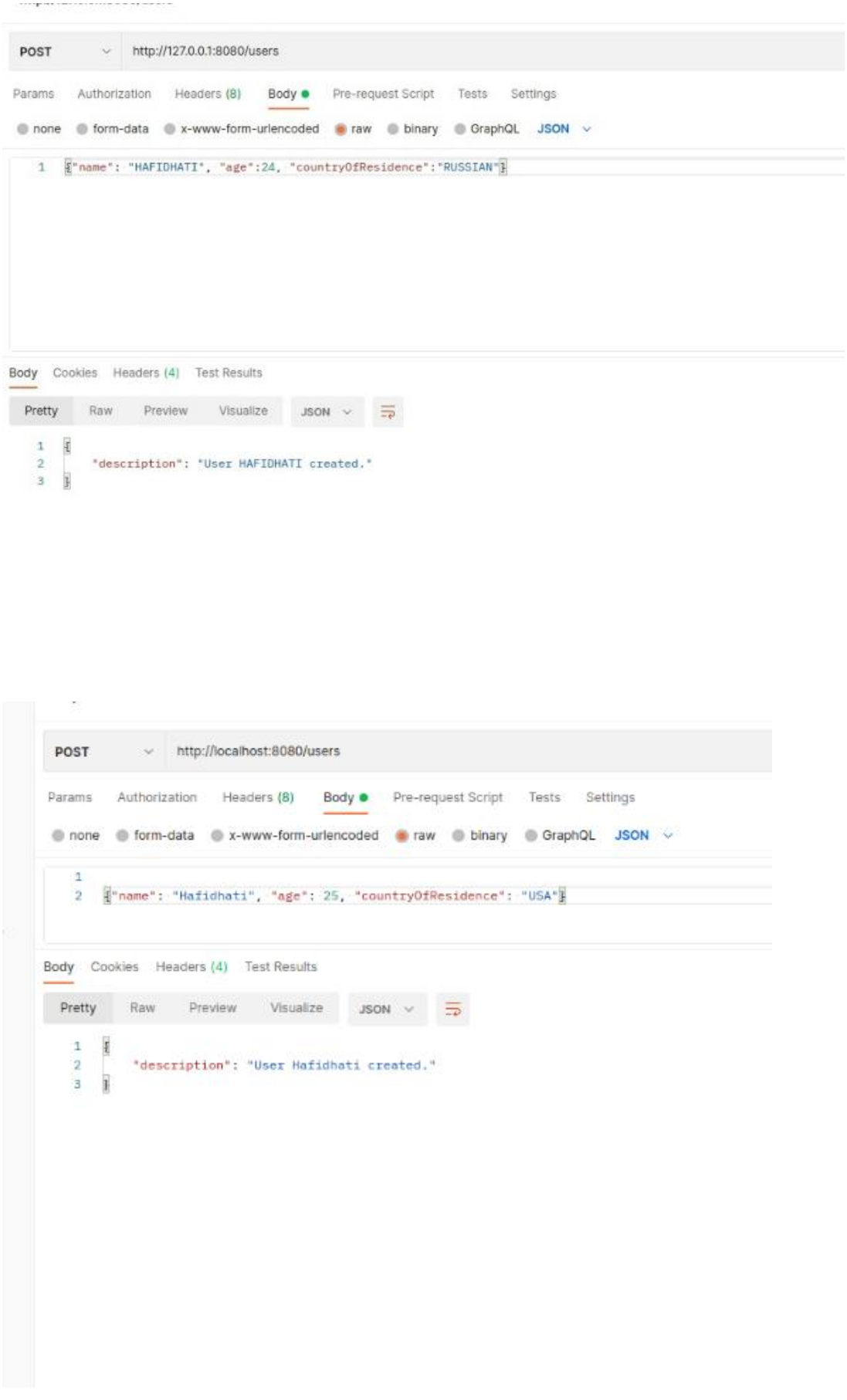


рисунок 10. С запросом отправленным на 8080 порт и ответом

Также мы можем получить список всех «Зарегистрированных» пользователей, для этого достаточно обратиться к серверу через GET запрос

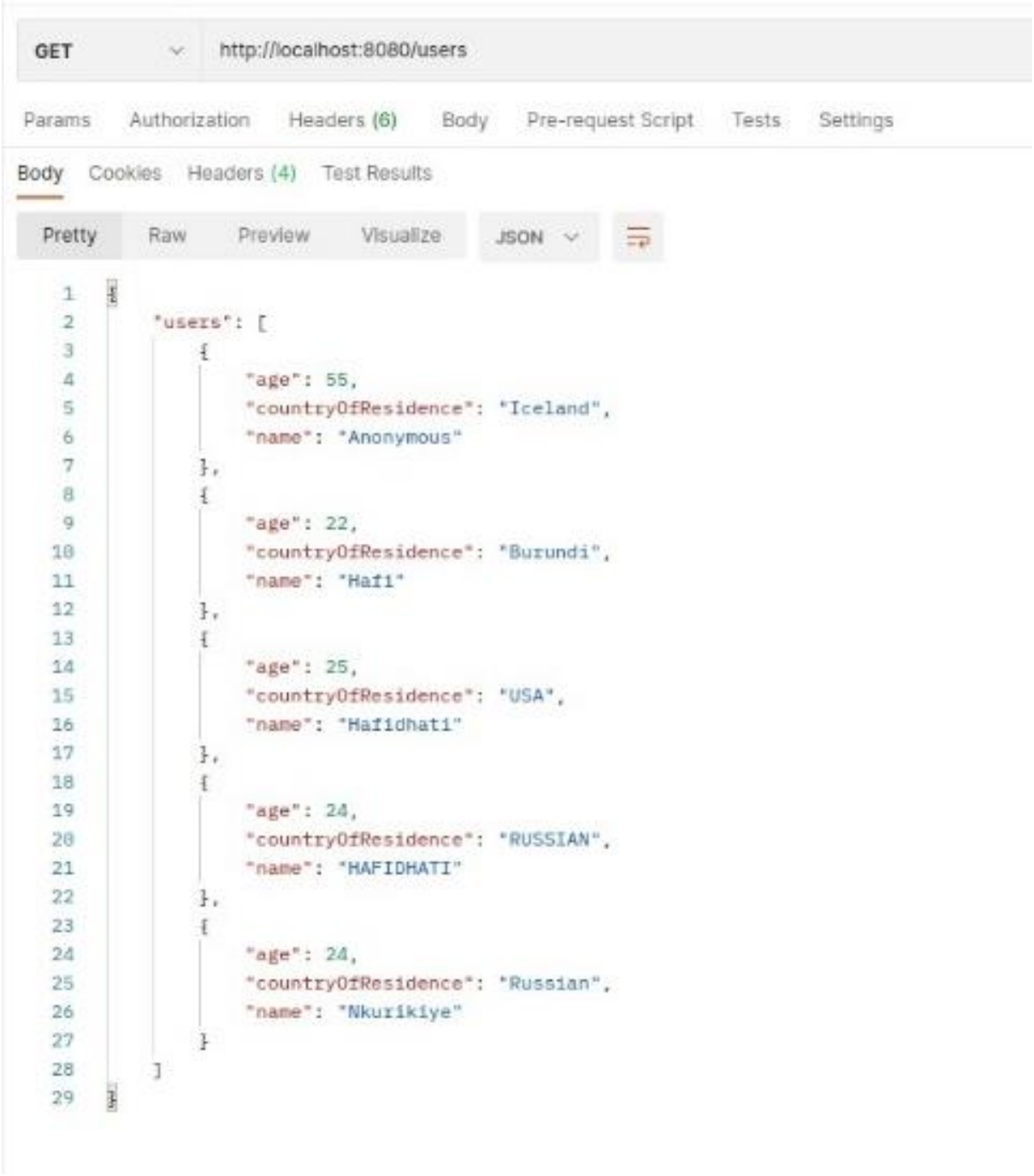


рисунок 11. Ответ сервера после «регистрации» еще нескольких пользователей, на запрос GET

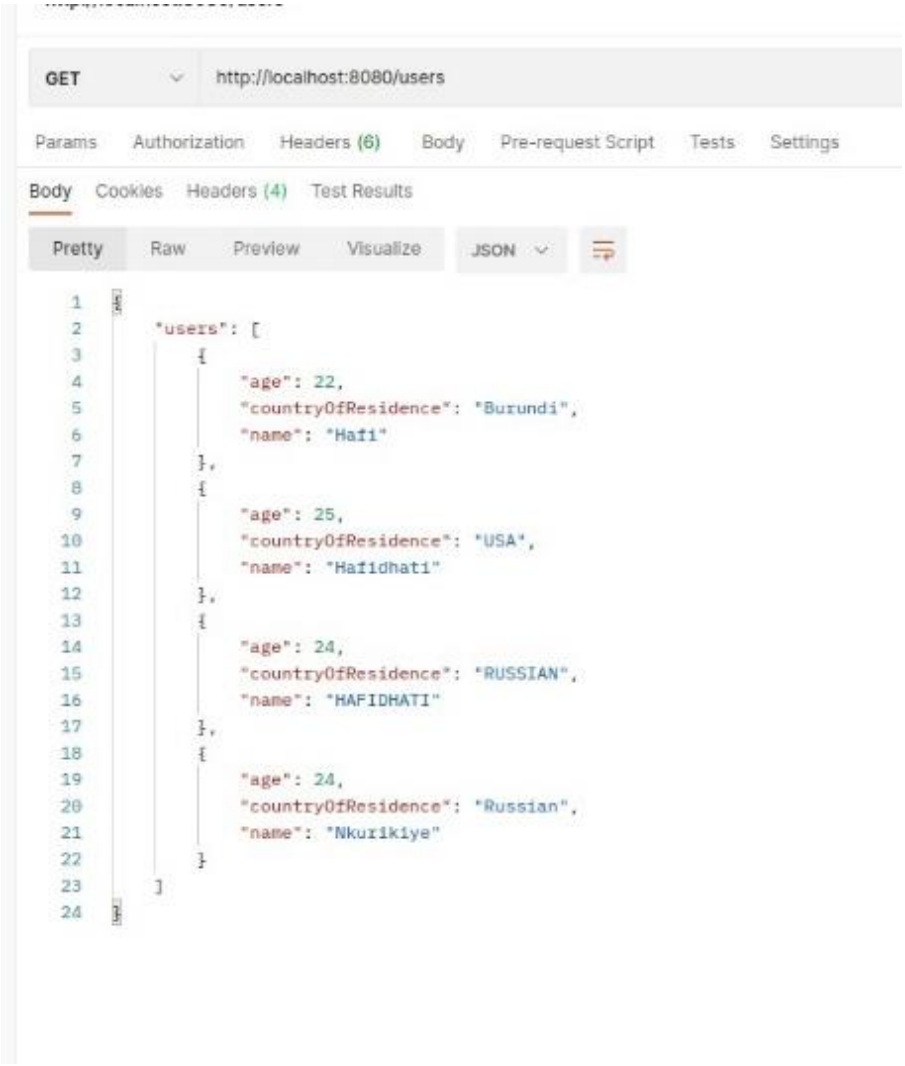
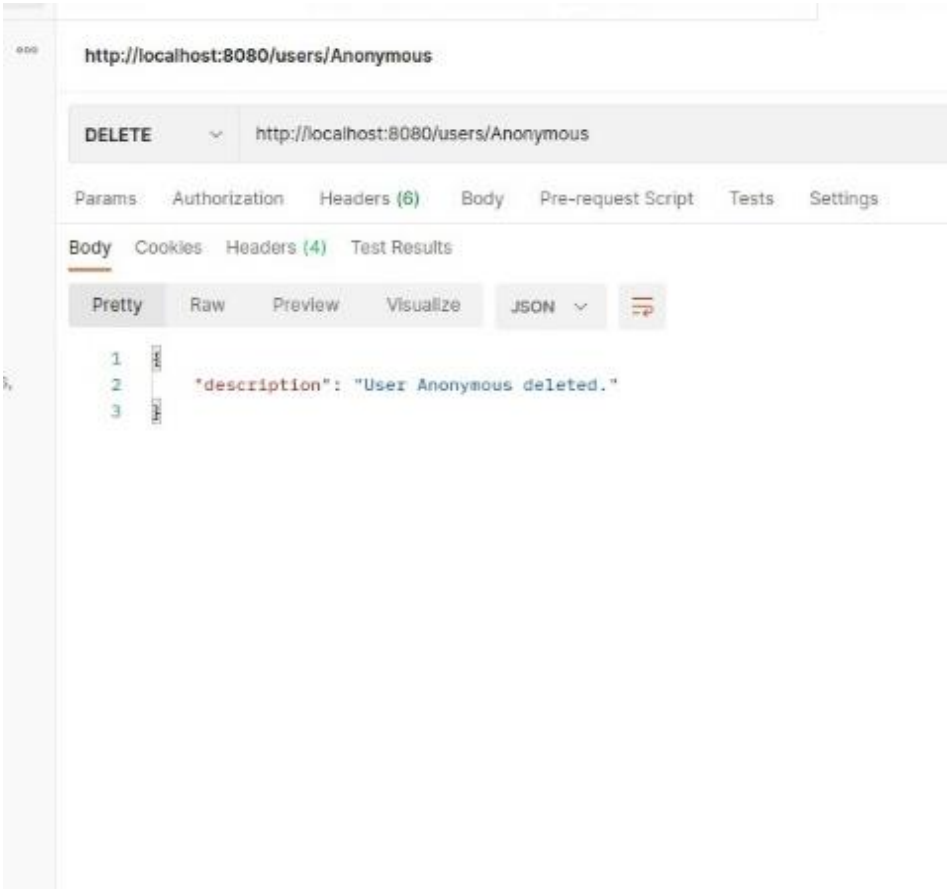


Рисунок 12. Получение информации по конкретному пользователю с помощью GET запроса

Также по мимо добавления новых пользователей и просмотра данных по уже созданным можно удалять пользователей обратившись к серверу с DELETE запросом и указанием имени пользователя.



Снимок 7. Результат удаления пользователя “Anonymous”.

Маршруты и их тестирование

Все маршруты выделены в отдельный класс. Это сделано для того, чтобы отделить код инфраструктуры от маршрутов, которые должны декларировать только то, с чем им нужно работать, и, следовательно, могут быть немного больше сосредоточены на своей задаче. Это, конечно, улучшает тестируемость, позволяя производить ее модульно.

Вывод

Данный проект позволил нам получить навыки работы с Akka.